PODSTAWY PROGRAMOWANIA W PYTHON

Dzień 5

ANKIETA 1

https://goo.gl/forms/BLWpBgwKMNRlO1Hm1

AGENDA
DAY 6

- Tuple
- Definiowanie funkcji
- Argumenty funkcji
- return
- docstring

1. Tuple

tuple
krotka ()

Tuple jest typem niezmiennym – raz zdefiniowanego nie można zmienić

```
tuple1 = ("raz", "dwa", "trzy")

tuple1[0] = "jeden" – spowoduje błąd

x = "raz",
y = "raz", dwa"
```

rozpakowanie tupli

```
tuple1 = ("raz", "dwa", "trzy")

x, y, z = tuple1

print(x)
>>> "raz"
print(y)
>>> "dwa"
print(z)
>>> "trzy"
```

for **indeks**, **element** in enumerate(kolekcja):

2.
Definiowanie funkcji

Tworzymy re-używalny kod

tworzenie funkcji

```
def do_nothing():
    pass
```

```
>>> do_nothing()
```

<- nic nie zrobił (dokładnie to co chcieliśmy)

```
>>> x = do_nothing
>>> x()
```

<- funkcja jest obiektem możemy więc zapisać do zmiennej

<- zmienna z funkcją można wywołać

tworzenie funkcji

```
1  give_square(35)
2
3  def give_square(x):
4      print(x**2)
5
```

NIE

```
give_square(35)
NameError: name 'give_square' is not defined
```

```
1  def give_square(x):
2      print(x**2)
3
4  give_square(35)
5
```

TAK

argumenty funkcji

```
def do_nothing():
    pass
```

nie ma argumentów

```
def do_nothing(x):
    pass
```

jeden argument

```
def do_nothing(x, y, z):
    pass
```

wiele argumentów

argumenty domyślne

```
def do_nothing(x, y=10):
    pass

def do_nothing(x, y, z=12, w =„Ola”):
    pass

def do_nothing(y=10):
    pass
```

argumenty domyślne muszą być po argumentach wymaganych

argument domyślny jest sprawdzany tylko przy pierwszym wywołaniu funkcji – uwaga na typy referencyjne!

argumenty domyślne

wywołanie funkcji

```
def do_something(x, y, z=12, w =„Ola”):
    pass

>>> do_something(1)        <- błąd – wszystkie arg. pozycyjne muszą być podane
>>> do_something(1, 23)
>>> do_something(1, 2, "trzy")
>>> do_something(1, 2, 34, "ola")
>>> do_something(1, 33, w="ola")
```

return

funkcja może robić coś wewnątrz siebie **(nawet nie trzeba print)**

```
def print_square(x)
    print(x**2)
```

funkcja może oddać jakiś wynik/obiekt – używamy return

```
def give_square(x)
    return x**2
```

aby użyć funkcję zwracającą obiekt należy ten obiekt zapisać w zmiennej

```
>>> wynik = give_square(3)
>>> print(wynik)
9
>>>
```

argumenty domyślne – typy referencyjne

```
def dodaj_imie(imie, imiona=[]):
    imiona.append(imie)
    return imiona

>>> print(dodaj_imie(„Ala”))
[„Ala”]
>>> print(dodaj_imie(„Ola”))
[„Ala”, „Ola”]
>>> print(dodaj_imie(„Ewa”))
[„Ala”, „Ola”, „Ewa”]
```

```
def dodaj_imie(imie, imiona=None):
    if imiona == None:
        imiona = []
    imiona.append(imie)
    return imiona

>>> print(dodaj_imie(„Ala”))
[„Ala”]
>>> print(dodaj_imie(„Ola”))
[„Ola”]
>>> print(dodaj_imie(„Ewa”))
[„Ewa”]
```

3.
Zakres zmiennych

Jak daleko sięgać mogę

ZAKRES ZMIENNYCH
zmienne lokalne i globalne

Zmienne lokalne funkcji są do wykorzystania tylko w tej funkcji (i głębiej)

Co jeśli mamy zmienną **imie** na poziomie głównego programu, oraz zmienną lokalną **imie** na poziomie funkcji

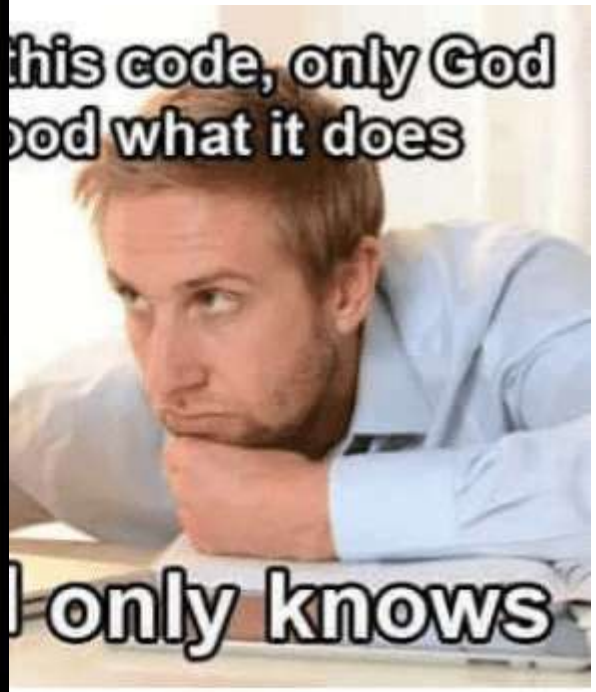
```
imie = "jola"

def drukuj_imiona(imie_2):
    imie = "ania"
```

4.
docstring

dokumentuj kod


"WHO WROTE THIS AWFUL CODE?!?" RIGHT CLICKS 'ANNOTATE'...
IT WAS ME.
memegenerator.net


his code, only God
od what it does
only knows

docstring – PEP257

documentation string

```
def do_nothing(x, y=10):
    """Does absolutely nothing"""
    pass


def give_square(x):
    """Returns square of given numer

    (numer) -> numer
    """
    return x**2
```

https://www.python.org/dev/peps/pep-0257/

Thanks!!